The Implementation of a COTS Based Fault Tolerant Avionics Bus Architecture

Eric Holmberg, Savio Chau, Huy Luong, William Charlan, Ryan Fukuhara,
Peter Jones, Gregory Pixler

Jet Propulsion Laboratory
4800 Oak Grove Drive,
Pasadena, CA 91109

Abstract

X2000 is a technology program at the Jet Propulsion Laboratory to develop enabling technologies for future flight missions at affordable cost. The cost constraints mandate the use of commercial-off-the-shelf (COTS) components and standards in the X2000 avionics system architecture. Furthermore, the X2000 avionics system design has to be applicable to multiple missions, so that non-recurring engineering costs can be shared by the missions.

The X2000 has selected two commercial bus standards, the IEEE 1394 and I^2C, as the avionics system buses. These two buses work together to provide the performance, scalability, low power consumption, and fault tolerance required by long life deep space missions. In this paper, we report our approach in implementing a fault-tolerant bus architecture for the X2000 avionics system using these two COTS buses. The system approach is described first. Then, the focus of the rest of the discussions are on the implementation of two ASICs, named Digital I/O (DIO) and Mixed Signal I/O (MSIO) ASICs, which are the key components of this COTS based fault-tolerant bus architecture.

1. Introduction

In recent years, commercial-off-the-shelf (COTS) products have found many applications in space exploration. The attractiveness of COTS is that low cost hardware and software products are widely available in the commercial market. By using COTS throughout the system, we expect to significantly reduce both the development cost as well as the recurring cost of the system. On the other hand, COTS are not specifically developed for highly reliable applications such as long-life deep-space missions. The real challenge is to deliver a low-cost, highly reliable

and long-term survivable system based on COTS that are not developed with high-reliability in mind. In this paper, we report our approach of using COTS to implement a fault-tolerant avionics system for the Advanced Deep Spacecraft System Technology Program (also known as X2000) at the Jet Propulsion Laboratory.

The X2000 avionics system design emphasizes an architectural flexibility and scalability, so that it can be reused for multiple missions in order to reduce the cost of space exploration. The X2000 architecture is a distributed, symmetric system of multiple computing nodes and device drivers that share a common redundant bus architecture. Most notably, all interfaces used in this distributed architecture are based on COTS. That is, the local computer bus is the Peripheral Component Interface (PCI) bus; the "system" bus is the high-speed IEEE 1394 bus; and the I^2C bus is used as a maintenance bus in the system level and as an engineering bus in the subsystem level (see Figure 1). The IEEE 1394 and the I^2C buses are multi-master and capable to support distributed systems. These two buses were selected after an intensive search and trade-off study conducted at the Jet Propulsion Laboratory in 1997.

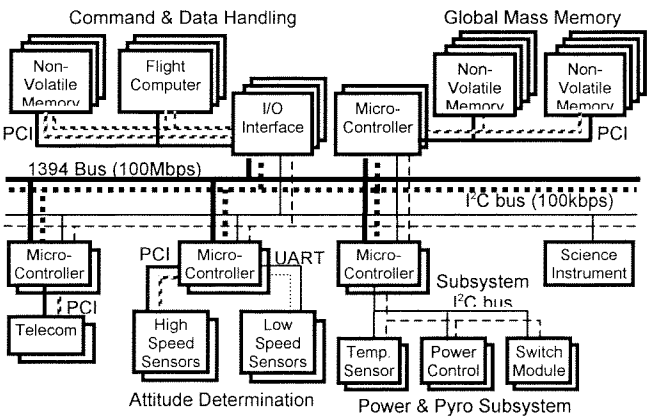

Figure 1: X2000 Avionics System Architecture

While the COTS buses have many benefits, their commercial products are not suitable for space applications for two reasons. First, the fault protection designs of these buses are insufficient to meet the reliability requirements of long life deep space missions. Second, these commercial electronic components do not meet the space environment requirements such as extremely high

or low temperatures, very low power consumption, and very high total dose and single event upset radiation tolerance, especially for those missions that require Jupiter gravity assist.

To solve the first problem, a multi-level fault protection design methodology has been adopted to overcome the shortcomings of the COTS buses [12]. First, the X2000 exploits the built-in fault detection mechanisms in the bus standards as the first line of defense. Second, an additional layer of hardware and software are used to enhance the fault detection, isolation, and recovery capabilities of each individual bus. This level requires substantial design and therefore is the main focus of this paper. Third, any single point failure in each bus is removed by mutual protection between the IEEE 1394 and I^2C. Finally, the IEEE 1394 and the I^2C buses are replicated at the system level to provide one more level of fault protection for long life missions.

As far as the second problem is concerned, one way to solve the problem is to redesign the components of the COTS buses and then fabricate them in a space-qualified ASIC foundry. While this approach can produce space-qualified parts, the development cost will be prohibitive if the redesign has to be done from the very beginning. This is because the ASICs to implement the COTS buses and the enhanced fault protection features are very complicated and involves more than a half million gates. Fortunately, as another benefit of using COTS bus standards, there are many commercial ASIC cores (referred to as Intellectual Properties or IPs) that have been designed for these buses and can be purchased off-the-shelf. The most important advantage of IPs is that they are reusable and can be integrated into a custom ASIC design, and fabricated by space-qualified foundry in a very short time. In the following, we will first describe the X2000 fault tolerant bus architecture. Then the discussion will be focused on how X2000 implements the fault tolerant bus architecture by developing ASICs based on the COTS IPs.

2. The X2000 Fault Tolerant Bus Architecture

The development of the X2000 fault tolerant bus architecture follows the aforementioned multi-level fault protection design methodology as follows.

First, the built-in fault detection capabilities such as the cyclic redundancy check (CRC), acknowledgment packets, response packet error codes, and time-out conditions of the IEEE 1394 bus [2][3] and the acknowledgment bit of the I^2C bus [4][5] are used as the first line of defense in fault detection. A very important capability in the revised version of the IEEE 1394 standard (IEEE 1394a [7]) is the ability of a node to enable or disable its individual ports (a port is the physical interface to a link). With this capability, a node in the bus can disable a link connected to a failed node and enable a backup link to bypass the failed node. This capability is the basis of the IEEE 1394 bus recovery in this bus architecture.

Secondly, additional hardware and software fault tolerance capabilities are used to detect faults that are not detectable by the built-in capabilities of the buses. These capabilities include a layer of fault detection message protocol, watchdog timers, etc. The most notable addition of such fault tolerance enhancement is the *fail silence protocol* of the I^2C bus. This protocol requires each flight computer or microcontroller to send a special message (called fail-silence message) to itself periodically. If the bus is shorted or one of the nodes is babbling, the fail-silence message will not be delivered in time. Upon failing to receive its own fail-silence message, the node will automatically shut off its bus transmitter. Eventually, the faulty node will shut itself off and the bus will be operational again. At this point, a designated master node will poll all other nodes individually to find out if they are shut off. If a node causes the bus to fail again when it is polled, then it will be identified as the faulty node and will be shut off permanently. If the master node is the faulty node, other backup master nodes can be invoked to carry out the recovery operation.

There are some failures that might be very difficult to recover from by the enhanced fault tolerance capabilities. For example, the tree topology of the IEEE 1394 bus can be partitioned by a node or link failure. Even though the fault can easily be detected, it is very difficult for the IEEE 1394 bus to recover from such fault by itself because the healthy nodes are not able to communication with each other to coordinate the fault recovery. For this reason, the third level of the fault tolerance design methodology calls for a mutual protection strategy between the IEEE 1394 and I^2C buses. For the example given above, the I^2C bus is used to facilitate the communication among the healthy nodes to identify the faulty node. Once it is identified, all

other nodes can disable their ports connecting to the faulty node and activate their backup connections to restablish the connectivity of the bus. This is illustrated in Figure 2.

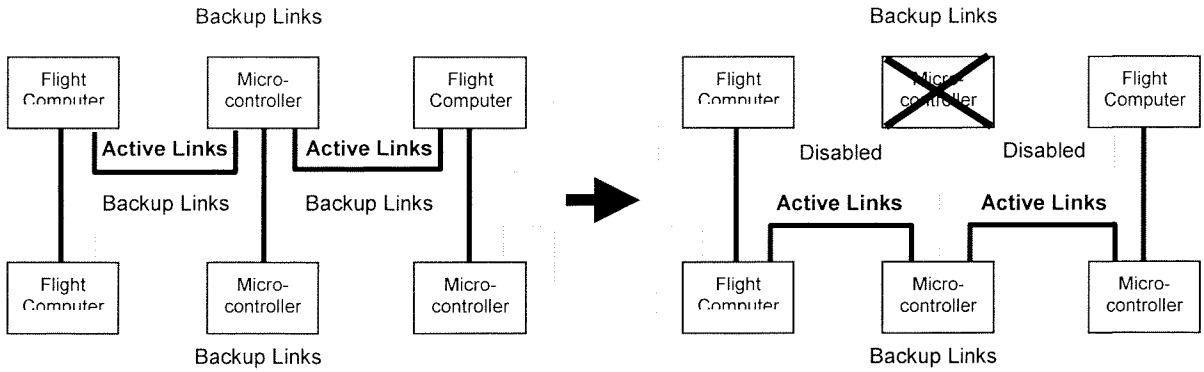

Figure 2: Fault Recovery of the IEEE 1394 Bus

Finally, the entire IEEE 1394 and I^2C bus set is duplicated to provide an additional system level of fault protection. In order to further enhance the effectiveness of the system level redundancy, the backup IEEE 1394 bus uses a slightly different active connectivity from the primary IEEE 1394 bus. The stipulation is that any branch node in the primary IEEE 1394 bus should not be a branch node in the backup IEEE 1394 bus and vice versa. Hence, when a node fails, it can only partition the bus in which it is a branch node. As far as the second bus is concerned, the failure only represents the loss of a leaf node, but the main body of the tree structure is not affected. This tatic leads to a "stack-tree" topology. Figure 3 illustrates an example implementation of the stack-tree topology. A model-based quantitative evaluation [10] shows that, for an 11-year mission and a node failure rate of 1×10^{-7} failures/hr, the system reliability of a 32-node instance of the COTS-based fault-tolerant bus architecture can remain greater than 0.9999 at the end of mission. In contrast, the reliability of a non-fault tolerant COTS-based bus of the same size and node failure rate is less than 0.86 at the end of a mission with the same duration.

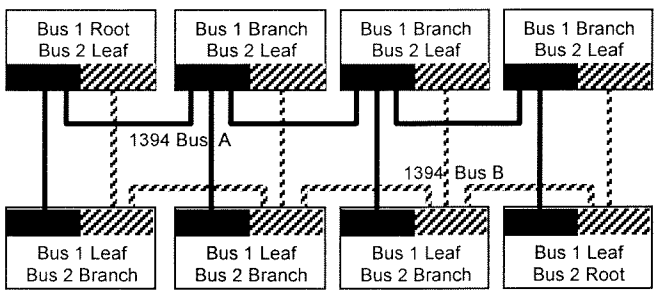

Figure 3: Stack-tree topology of IEEE 1394 bus
(Note: backup connections are not shown)

3. Design of the Fault Tolerant Bus Interface

The X2000 fault protection avionics bus architecture is implemented by a system I/O assembly referred to as the SIO slice, which is depicted in Figure 4. Each flight computer or microcontroller has to interface with the IEEE 1394 and I^2C buses through the SIO slice. The main function of the SIO slice is to interface the system buses (i.e., IEEE 1394 and I^2C) to the internal PCI bus in the flight computer or microcontroller. The SIO slice consists of two redundant bus interface units, each of which has a 3-port IEEE 1394 bus interface and two I^2C bus interfaces. The bus interface unit is composed of two ASICs: the Digital I/O (DIO) ASIC and the Mixed Signal I/O (MSIO) ASIC. The DIO ASIC implements the Link layer of the IEEE 1394 bus, the two I^2C bus controllers, and the logic for fault tolerance enhancements of the buses. In addition, the DIO ASIC also includes functions such as the UART to support software development and discrete output signals for miscellaneous control. The MSIO ASIC implements the Physical layer of the IEEE 1394 bus and the bus drivers of the I^2C buses. The MSIO is electrically isolated from the DIO and the rest of the flight computer or microcontroller. This is to prevent the propagation of electrical faults between the bus and the nodes and to mitigate grounding problems.

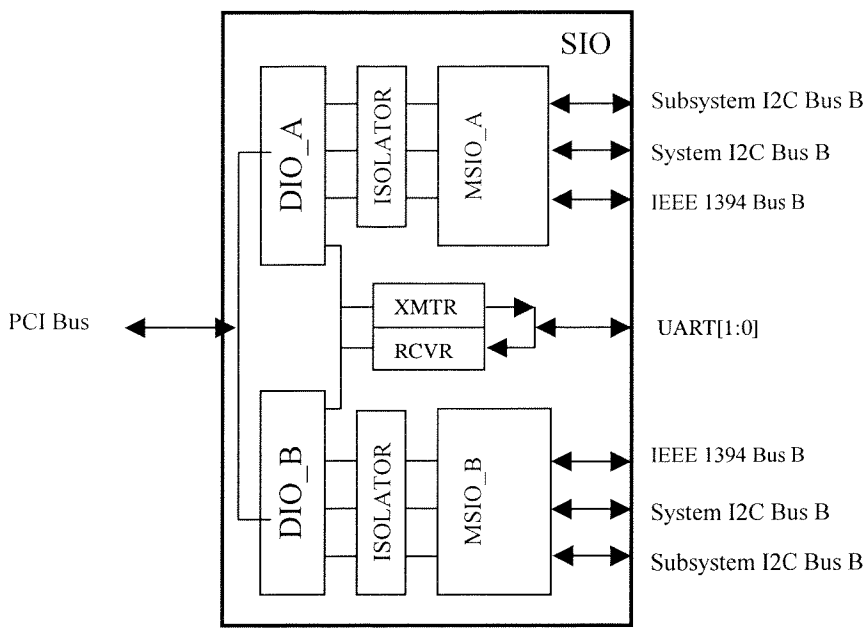

Figure 4: System I/O Slice

3.1 Description of the DIO ASIC

PCI Interface: The Digital Input/Output (DIO) ASIC is a bridge between the IEEE 1394/I^2C buses and the internal Compact PCI (cPCI) bus in a node (Figure 5) [11]. From the cPCI bus point of view, the DIO ASIC is single function bus device, using five of the standard six base addresses: 1394 bus Link layer controller, System I^2C bus controller, Subsystem I^2C bus controller, a UART and some custom logic. All of the bus controllers and the UART are COTS Intellectual Properties. The DIO ASIC has some custom logic to handle fault protection, timing control and reset.

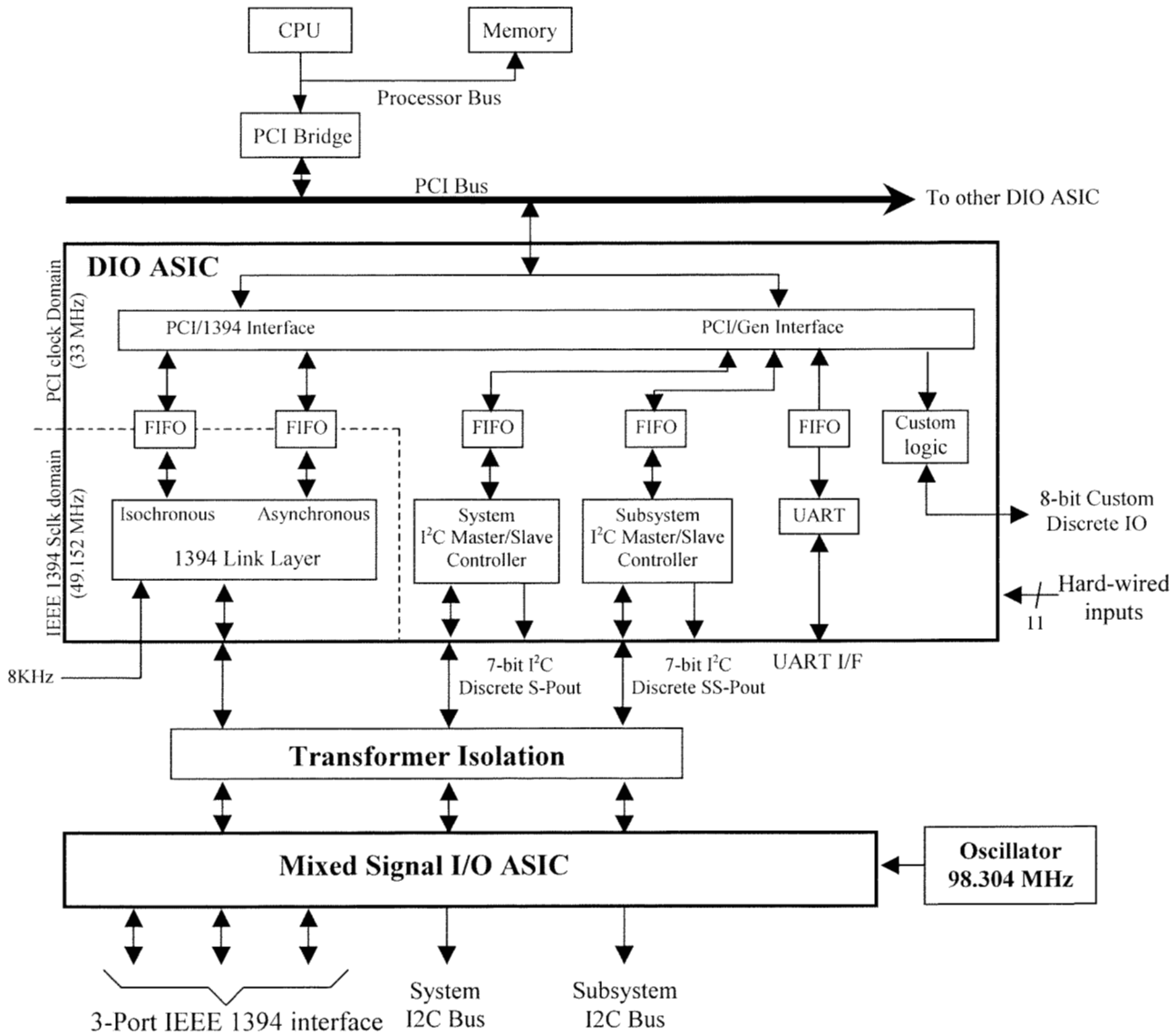

Figure 5: DIO and MSIO ASIC Design

The cPCI bus is a 32-bit bus operating at 32.768 MHz and compliant with the version 2.1 specification. Both initiator and target capabilities are implemented by the DIO ASIC. There are two types of target interfaces, register and memory. The register interface does not provide burst transfer, while the memory interface supports burst transfer with no wait-states. The registers and memory of the DIO occupy 384 Kbytes of the PCI bus address space, not counting the 64 bytes for the PCI configuration registers.

IEEE 1394 Bus Interface: From the IEEE 1394 point of view, the DIO ASIC is the link layer controller that handles both the isochronous and asynchronous transactions. The link layer controller IP supports the standard data rates of 100 Mbit/sec, 200 Mbit/sec, and 400 Mbit/sec. (Note: only the 100 Mbit/sec data rate is supported by MSIO.) The isochronous transactions guarantees on-time delivery by sending data in regular intervals called isochronous cycles, while the asynchronous transaction guarantees reliable delivery by message acknowledgment. Since the isochronous cycle has a period of 125 μs, the flight computer (or microcontroller) can potentially be interrupted 8000 times a second. In order to alleviate such burden from the flight computer, a direct memory access (DMA) engine is built into the DIO ASIC. All IEEE 1394 bus transactions, transmit and receive, asynchronous and isochronous, are carried out through DMA operations. The processor only has to set up DMA memory address locations and ranges. Afterwards, data from the IEEE 1394 bus will be put into the memory automatically. When the IEEE 1394 bus transaction is complete, the DIO ASIC will notify the flight computer through interrupts.

In addition to the standard isochronous transmissions, the DIO ASIC has also been added the capability of skipping cycles. Hence, a node can begin transmitting a block of data on a particular isochronous cycle and then skip every N cycles before the next transmission. This allows a number of nodes to transmit on the same channel when there is no need for any single node to transmit every cycle.

I^2C Bus Interface: From the I^2C bus's point of view, the DIO ASIC is the bus controller. The COTS I^2C controller IP has been enhanced to support fault tolerance, such as watchdog timers and a message level protocol that includes Cyclic Redundancy Code (CRC) generation and

checking. All messages, transmitted and received, are time stamped. The I^2C controller has two hardware selectable signaling rates, 100 kHz and 400 kHz. (Note: only the 100 kHz signaling rate is supported by the MSIO.)

There are two I^2C bus controllers in a DIO ASIC, one for a subsystem bus and one for a system bus. Both the subsystem and system I^2C buses have a shadow controller IP that facilitates the fail silence function for the flight computer or microcontroller nodes. The shadow controller IP can receive but not transmit messages. The shadow controller IP has an address equal to the primary node address plus one. This shadow controller is necessary because the COTS I^2C IP is not able to acknowledge messages address to itself, and thus cannot implement the fail silence protocol directly. With the shadow controller that has a slightly different address from the primary address, a node can send the fail silence message back to itself without violating the protocol.

As mentioned earlier, the fail-silence protocol requires every flight computer (or microcontroller) to send the fail-silence message back to itself periodically. The fail-silence message is sent via the nodes' primary address to its shadow address. Upon the receipt of the message, the flight computer resets a watchdog timer, which monitors the interval between the fail-silence messages. If the watchdog timer expires, a hardware signal is sent to the MSIO, shutting off the bus transmitter and stops all data transmissions. There are simple nodes on the bus that do not have the capability to send the fail-silence message to itself. In that case, their shadow controller of the DIO ASIC will be disabled by an input pin, but the master node of the I^2C bus will send the fail-silence message to reset the watchdog timer of these nodes.

While the bus transmitter is being shut off, the bus receiver of the node is still operational. Therefore, when the bus master polls the node, it is able to receive the polling message and re-enable its bus transmitter. If the node is the faulty one that caused the I^2C bus to enter the fail-silent state, it will cause the bus to fail again when its bus transmitter is re-enabled. Thus, the master is able to recognize this is a faulty node and will not enable it again in future bus recovery. On the other hand, if the master node is the faulty node, this recovery scheme may fail. Another master node will have to be elected through the IEEE 1394 bus.

UART Interface: The DIO ASIC has a 16550 UART interface to support software development or to be used as a dedicated peripheral interface. The UART has programmable baud rate from 50 to 512 Kbauds. The UART interface shares the PCI bus interface with the IEEE 1394 and the I^2C bus controllers but used only as a target on the PCI bus. When the flight computer sends data to the UART, it sends the data in a 32-bit word (referred to as a quadlet) format across the PCI bus to the DIO ASIC. The 32-bit quadlet will be stored in a FIFO and then converted to the 8-bit serial format of the UART protocol. When data is received from the UART interface, the DIO ASIC concatenates the 8-bit data bytes into 32-bit quadlets so that four bytes can be transferred on the PCI bus simultaneously.

The UART has a Transmit FIFO and a Receive FIFO. Each FIFO is divided into two 32-bit wide and 64-quadlet deep sections. The two sections operate in a ping-pong fashion, in which the UART can access one section while the software, through the PCI bus, can accesses the other. The Transmit FIFO can be bypassed if single bytes are transmitted.

The registers in the UART interface are accessible by the flight computer through the PCI bus. There are two sets of UART registers. The first set of registers are one byte-wide and compatible with National's PC16550D, while the other set are customized 32-bit wide registers for FIFO control and status.

Custom Logic: The Custom Logic also shares the PCI bus interface with the other bus controllers and is also used only as a target on the PCI bus. The Custom Logic performs eight major functions:

- Control for eight types of reset
- Interrupt logic: "ORing" of all interrupts, interrupt and mask registers
- Self-test watchdog timer: initiates a self-test procedure that requires three successive writes of correct patterns to three specific memory locations to verify the health of the flight computer.
- An array of four general purpose timers

- Clock controller/generator: generates the required clocks for all function blocks in the DIO ASIC and enable/disable clocks for power management
- Node identification logic
- Test controller: Boundary scan, internal scan and a timer scan

3.2 Description of the MSIO ASIC

The Mixed-Signal Input/Output (MSIO) ASIC has a 3-port IEEE 1394 S100 (100 Mbit/sec) cable Physical layer (Phy) and transceivers for two I^2C buses (100 kbit/sec) [13]. Both the I^2C circuitry and IEEE 1394 circuitry interface to the DIO via transformer isolation. The IEEE 1394 Phy portion of the MSIO performs the standard functions of bus configuration, node insertion/removal detection, bus arbitration, physical connection to other nodes, encoding and decoding, synchronization and resynchronization. In addition, enhancements in the revised version of the standard (i.e., IEEE P1394a) such as port disable are also supported.

The I^2C bus is a multi-drop bus, using passive pull-ups on single ended (unbalanced) output drivers. The output driver (i.e., the bus transmitter) is essentially a pull-down transistor. Since the bus uses passive pull-up and active pull-down, a logical 0 on the bus always wins over a logical 1. This characteristic is used in bus arbitration in the following way. Any node transmitting data has to monitor the logic level on the bus with a bus receiver, which is compared with the sent data. If the bus receiver output disagrees with the sent data, the node has to stop the data transmission. Otherwise, the data transmission can continue. The MSIO has a field effect transistor (FET) put in series with the pull-down transistor of the output driver. The FET is controlled from the DIO and is used to disable the output driver when the node is fail-silent. There are also outputs allowing an external fail-silent switch to be used in series with the pull-down transistor of the output driver.

The standard mode (100 kHz) in the I^2C specification allows a maximum bus capacitance of 400 pf and a minimum bus driver sink current capability of 3 mA. To meet the 1 us maximum rise time requirement, the 3 mA minimum sink current limits the bus length to less than 7 ft. This is

because the bus was intended to be contained within a chassis. To avoid transmission line effects, with a fast fall time, the bus has to be less than 7 ft. However, some I^2C users at JPL need a much longer bus, up to 20 meters. There is also a requirement to support 60 nodes on the bus. To accommodate these requirements, the bus driver's sink capability has been increased to 30 mA and slew rate control has been added to drivers. The maximum slew rate has been set so that the 20-meter bus does not appear as a transmission line.

3.3 Description of the Isolator

As the nodes in the spacecraft may have different grounds and some nodes such as the pyrotechnic may generate large transients, it is necessary to isolate each node electrically from the bus media. In X2000, the isolation is inserted between the DIO and MSIO ASIC via transformers [14]. Since transformers cannot relay DC signals, both the DIO and MSIO ASICs have to modulate DC signals into pulsed signals in order to pass through the transformers. The isolation circuit is shown in Figure 6.

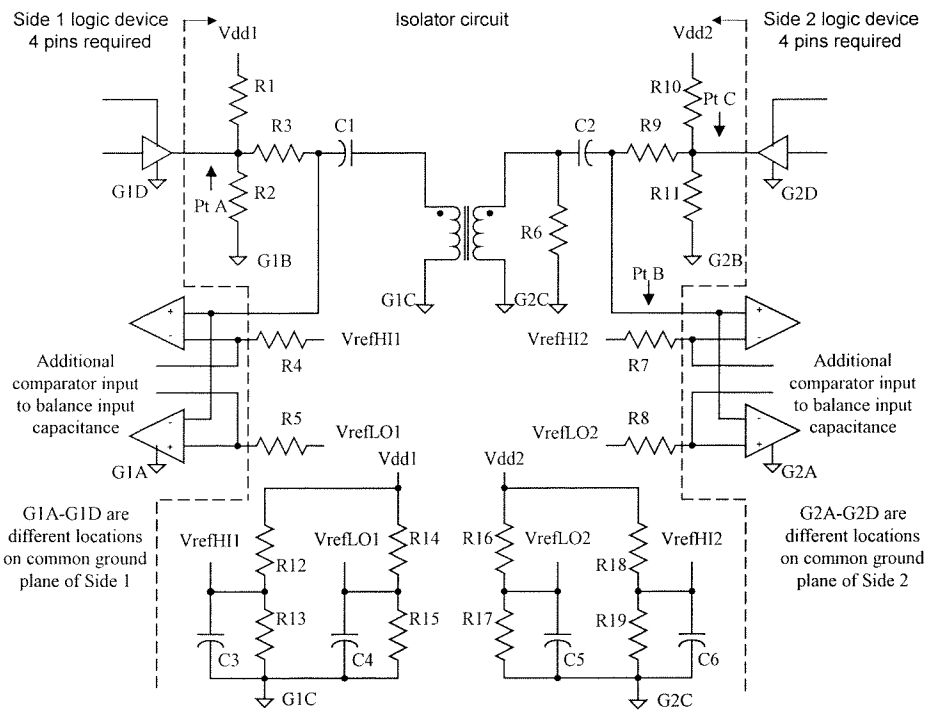

Figure 6: DIO and MSIO ASIC Isolation Circuit

4. Status of the DIO and MSIO Development

4.1 Status of the DIO Development

The design of the DIO ASIC is underway. JPL has procured the intellectual properties (IP) for the majority of the DIO design. Since the IP cores do not fully meet JPL's requirements, an outside contractor has been working along with JPL to design the needed enhancements.

In order to reduce the technical risk, it was decided to implement the DIO design on a prototype board using Xilinx FPGA's before the ASIC is fabricated. The prototype board uses a Texas Instruments' commercial IEEE 1394 Phy chip, but due to speed limitations of the FPGAs, runs at reduced speed. The board will be tested at the contractor's facility and will be shipped to JPL in December of 1999. JPL will continue the testing with the prototype board in an in-house facility.

The DIO ASIC will be fabricated on the Honeywell HX3000 line, with the first ASIC delivered to JPL in early April of 2000.

4.2 MSIO Status

JPL has procured the digital part of the IEEE 1394 Physical Layer IP and is having a contractor design the analog circuitry. The MSIO ASIC will be fabricated using the Honeywell HX2000 library since the HX3000 line is relatively new and doesn't have a mixed signal cell library at this time.

JPL is currently working on the integration simulation of the analog and digital designs. The first ASIC is scheduled for delivery in the first part of May 2000.

5. Summary and Conclusion

In this paper, we have reported our approach to implementing a fault-tolerant bus architecture for X2000 avionics system using two COTS buses, the IEEE 1394 and the I^2C. On the system level, the IEEE 1394 or I^2C buses are duplicated but only one set of buses is active. When the active bus set fails, the normal operations will first be moved to the backup bus group. Then the failed bus set will be diagnosed, recovered, and used as backup bus for the next bus failure. The IEEE 1394 and I^2C buses work together to provide the fault isolation and recovery capabilities need for repairing the failed bus. This approach can provide much higher reliability for ultra long-life missions than the traditional dual redundancy approach.

Furthermore, this paper has provided detailed descriptions of two key components in the fault tolerant bus architecture: the DIO and MSIO ASICs. The DIO implements the Link layer of the IEEE 1394 bus, the I^2C bus controller, and a PCI bridge, and the MSIO ASIC implements the I^2C bus transceiver and the physical layer of the IEEE 1394 bus. The ASICs have additional logic circuits to facilitate the system level fault tolerance provisions. Examples of these fault tolerance provisions include the capability to reconfigure the IEEE 1394 bus topology to remove any failed node or links and a fail-silence protocol to check the health of the I^2C bus and initiate fault recovery. The DIO and MSIO development is underway at the Jet Propulsion Laboratory. Prototype level testing will be conducted within one year and the test results will be reported in future papers.

Acknowledgement

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

[1] L. Alkalai, "NASA Center for Integrated Space Microsystems," in *Proceedings of Advanced Deep Space System Development Program Workshop on Advanced Spacecraft Technologies*, (Pasadena, CA), Jun. 1997.

[2] IEEE 1394, *Standard for a High Performance Serial Bus*, Institute of Electrical and Electronic Engineers, Jan. 1995.

[3] D. Anderson, *FireWire System Architecture, IEEE 1394*. PC System Architecture Series, MA: Addison Wesley, 1998.

[4] D. Paret and C. Fenger, *The I^2C Bus: From Theory to Practice*. John Wiley, 1997.

[5] *The I^2C-Bus Specification, Version 2.0*. Philips Semiconductor, Dec. 1998.

[6] T. Shanley and D. Anderson, *PCI System Architecture*. Addison Wesley, 1995.

[7] IEEE P1394A, *Standard for a High Performance Serial Bus (Supplement)*, Draft 2.0. Institute of Electrical and Electronic Engineers, Mar. 1998.

[8] J. Marshall, "Building standard based COTS multiprocessor computer systems for space around a high speed serial bus network," in *Proceedings of the 17th Digital Avionics Systems Conference*, (Bellevue, Washington), Nov. 1998.

[9] J. Donaldson, "Cassini Orbiter Functional Requirements Book: Command and Data Subsystem," *JPL Document CAS-4-2006*, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, June 28, 1994

[10] A. T. Tai, S. N. Chau, and L. Alkalai, "COTS-based fault tolerance in deep space: qualitative and quantitative analyses of a bus network architecture," in *Proceedings of the 4th IEEE High-Assurance System Engineering Symposium,* (Washington, D.C.), Nov. 1999.

[11] H. Luong, "X2000 First Delivery Digital Input and Output (DIO) ASIC," *JPL Document D-16931*, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Mar. 1999.

[12] S. N. Chau, and L. Alkalai, A. T. Tai, and J. B. Burt, "The design of a fault-tolerant cots-based bus architecture," in *IEEE Transaction on Reliability*, Dec. 1999.

[13] P. Jones, "X2000 Integrated First Delivery Project Mixed- Signal Input/Output (MSIO) ASIC Requirements," *JPL Document D-18050, Rev 1.07*, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Aug. 1999.

[14] G. Pixler, "X2000 MSIO/DIO Transformer Interface Isolator Specification," *JPL Document*, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Aug. 1999.